Abstract Software Migration Architecture Towards Agent Middleware Interoperability

Cucurull, J.; Overeinder, B.J.; Oey, M.A.; Borrell, J.; Brazier, F.M.

published in
Proceedings of the 2nd Int'l Multiconference on Computer Science and Information Technology (IMCSIT)
2007

document version
Publisher's PDF, also known as Version of record

Proceedings of the International Multiconference on
Computer Science and Information Technology pp. 27–37
ISSN 1896-7094

Abstract software migration architecture towards agent middleware interoperability

Jordi Cucurull[1], Benno J. Overeinder[2], Michel A. Oey[2], Joan Borrell[1], and Frances M.T. Brazier[2]

[1] Department of Information and Communications Engineering,
Universitat Autònoma de Barcelona,
08193 Bellaterra, Spain
{jcucurull,jborrell}@deic.uab.cat
[2] Department of Computer Science, Vrije Universiteit Amsterdam,
Amsterdam, The Netherlands
{bjo,michel,frances}@cs.vu.nl

Abstract. Agent mobility is the ability of an agent to migrate from one location to another. So far, there are several difficulties with agents' migration due to the lack of interoperability among agent middleware that is distributed over the net. In this paper, an abstract software migration architecture is presented, which is the first step towards full agent middleware interoperability. With this architecture, the process of migrating an agent is uniformly defined for multiple middleware, leaving the agent execution environment' standards as a future research. To validate the suggested abstract migration, the architecture has been successfully implemented over two different agent middleware: JADE and AgentScape.

Keywords: Mobile Agents, Interoperability, IEEE-FIPA, migration, abstract implementation, JADE, AgentScape.

1 Introduction

The ability of an agent to migrate from one location to another provides a new dimension to multi-agent systems. An agent can decide to which location it chooses to migrate on the basis of the tasks it has at hand and the resources required. Different locations provide different services and different resources at different points in time.

Mobility between locations, however, requires interoperability between locations. Interoperability can be realised at different levels of a system: from operation systems level to application level. This paper proposes interoperability at application level. In this paper, an abstract software architecture is proposed to implement the Cucurull *et al.* [4] migration model based on a number of IEEE FIPA specifications. The model has been designed to support multiple migration protocols, providing developers the flexibility needed at application level.

To demonstrate the feasibility of the migration model, and the architecture design and implementation, two reference implementations of the architecture for the JADE [2] and AgentScape [10] agent middleware have been realised. Given the implementation

of the migration architecture on both middleware, agents can be sent between the two heterogeneous middleware types, but executing the agent is not possible (yet). The software migration architecture presented in this paper is, then, the first step towards a full agent middleware interoperability.

The paper is organised in seven sections: Section 2 summarises the migration architecture proposal; Section 3 presents the software migration architecture and its requirements; Section 4 presents the implementation of the architecture for the JADE platform; Section 5 describes the implementation of the architecture for the AgentScape platform; Section 6 states the related work; and Section 7 discusses the results and conclusions.

2 Migration proposal

2.1 Overview

The main concepts of the migration model proposed by Cucurull *et al.* [4] are presented in this section.

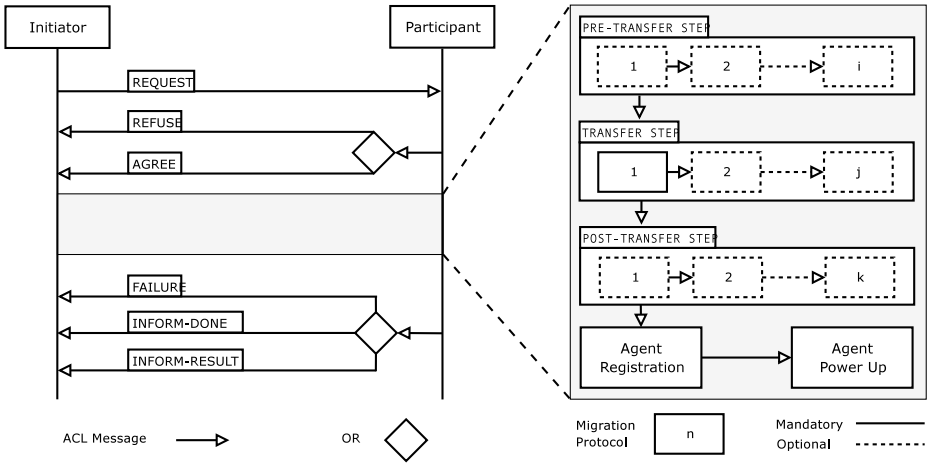

Fig. 1: Migration model.

The agent migration process is composed of a *Main Protocol* (left side of Figure 1), based on an IEEE-FIPA Request Interaction Protocol [6], and five steps within it (right side of Figure 1):

- **Pre-Transfer step**: Operations that need to run before agent transfer, e.g., authorisation, resource agreement, etc.
- **Transfer step**: Transfering of the agent code, data, and state. Different migration strategies (push, on demand, etc.) can be implemented.
- **Post-Transfer step**: Operations that need to run after agent transfer, e.g., authorisation, agent results transfer, etc.

– **Agent Registration step**: The agent is registered at the destination platform and, then, deleted at the origin (this prevents two agents running simultaneously and possibly get into conflict).
– **Agent Power Up step**: The agent is started in the destination platform.

The individual steps in the migration process are defined independently of each other. Each step in the protocol has its own (set of) protocols to perform its task. These protocols are well-defined in their functionality and operation, related to the step in the protocol; are independent of the other ones; can be composed of many stages; should use one or more ontologies and interaction protocols; and must have a unique name.

The first request–reply pair sent in the migration process follows an specific ontology specified in Cucurull *et al.* [4], and includes agent information related to its identity, its interoperability, and the protocols to use within the process. These protocols are the ones selected for the first three steps (Pre-Transfer, Transfer and Post-Transfer), which can be chosen by the agent before starting the migration process. The last two steps are fixed and always use the Agent Registration Protocol and the Agent Power Up Protocol respectively. If the two parts do not agree which protocol to use, or if some error happens during the process, the whole migration process is cancelled (a failure message is sent within the *Main Protocol*), and the agent execution is resumed at the local platform.

To have a minimum functional migration system, at least one protocol has to be available to the Transfer step. Three protocols are provided to illustrate the functionality of the model: the Push Transfer Protocol, the Agent Registration Protocol, and the Power Up Protocol. Note, however, that the Push Transfer Protocol is only an example. The choice of protocols for the first three steps is completely open. Developers are free to support the protocols of their choice—no default protocols are provided except for the last two steps.

2.2 Protocols

Push Transfer Protocol is the protocol used in the Transfer step to send all the agent code along with agent data and/or state within one message. Note that agent code is only sent if it is not already present in the remote middleware platform. The protocol is based on the use of the IEEE-FIPA Request and Proposal interaction protocols. The code, data and state sent are packed and coded according to the middleware platforms' requirements specified in the first message sent by the *Main Protocol*.

Agent Registration Protocol is the protocol used in the Agent Registration step. The protocol first sends a request to the remote middleware platform to rebuild and register the transmitted agent to the remote AMS. The AMS is a FIPA agent that maintains the directory of agents of each platform. If the remote middleware platforms acknowledges receipt, the agent's code, data, and state on the source middleware platform are deleted. This protocol uses an IEEE-FIPA Request Interaction protocol [6].

Agent Power Up Protocol is the protocol used in the Agent Power Up step. It requests the remote middleware platform to initiate the agent registered in the previous protocol. This protocol also uses an IEEE-FIPA Request Interaction protocol [6].

3 Proposed architecture

This section proposes an abstract software architecture to support the implementation of the migration model described in the previous section on the basis of requirements imposed to it. As stated earlier this architecture targets implementation of a migration protocol at application level, providing flexibility for developers.

3.1 Middleware requirements

The migration proposal outlined in Section 2 relies on the use of IEEE-FIPA specifications for the implementation of the different protocols in the five steps of the migration process. The agent middleware platform for which the implementation is targeted, should be FIPA-compliant or support at least:

- FIPA Agent naming scheme [7] or have a method to translate names.
- Agent directory service to register incoming agents.
- ACL messages since the whole migration architecture is based on exchanging them.
- Agent Communication Channel (the FIPA messaging service).
- FIPA SL content language to encode data included in the ACL messages.
- FIPA Request Interaction Protocol and other used within the migration protocols.

In addition to the requirements introduced above, certain rights to perform specific actions on the agent middleware platform are needed to perform migration of agents. For example, to send an agent to another agent middleware platform, the protocol needs access to the source middleware platform to access agent code, data, and state. Rights are needed to perform agent creation, to recreate and reinstantiate the migrated agent at the destination middleware.

3.2 Software architecture

Figure 2 shows the general idea of the application-level migration architecture, where each agent middleware platform has its own migration service. This service is provided by a dedicated agent, called the Agent Mobility Manager (AMM). The AMM agent implements the proposed migration protocol and communicates with the remote AMMs using ACL messages. As can be seen in Figure 2, the migration process is initiated by a message sent to the AMM (not defined by the proposal of Section 2 yet). This model supports migration initiated by the user and by the platform. Furthermore the AMM agent can host other future mobility related facilities, like acting as a message proxy for migrated agents, etc. Finally, the migration process supported by the architecture is:

1. Agent contacts local AMM, indicating it wants to migrate.
2. Local AMM suspends agent and gets the agent's code, data, and state.
3. Local AMM contacts remote AMM and starts migration:
   - pre-transfer step;
   - transfer step (local AMM sends agent code+data+state to remote AMM);
   - post-transfer step.

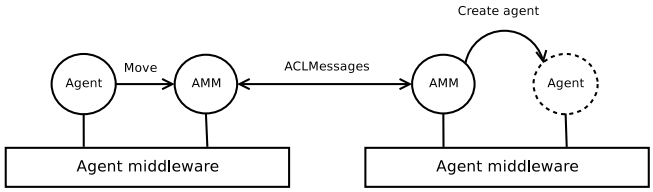

Fig. 2: Application level agent migration architecture.

4. Remote AMM creates and registers the new agent using the code+data+state (agent registration step).
5. Local AMM shuts down the local agent.
6. Remote AMM starts the new remote agent (agent power-up step).

The AMM should be supported by an agent middleware platform with the functionality described in the FIPA Agent Management specification [7]. The AMM is an extension of the FIPA architecture model. The internals of the AMM are left open to developers for each implementation.

4 JADE implementation

Most of the middleware requirements defined in Section 3.1 relate to the IEEE-FIPA specification. In this respect, JADE [2] provides all required functionality for the implementation of the migration architecture because it is FIPA compliant (see also [4]). This section presents the implementation of the migration protocol in JADE, making use of the service architecture of the JADE middleware. The section starts by describing this middleware, followed by the migration architecture implementation.

4.1 JADE middleware

Java Agent DEvelopment Framework (JADE) middleware [2] is a Java agent platform which provides a fully distributed system, where agents can reside on different hosts, an efficient transport mechanism with asynchronous messages, and internal (within a single JADE middleware platform instance) agent mobility support. An important feature of JADE is support for ontologies and content languages used in the agent messages. Each JADE agent runs as an independent *thread* and executes *behaviours*. Behaviours are tasks that can be dynamically inserted into and scheduled by an agent.

The JADE middleware is organised according to the FIPA specifications [7]. It is composed of the Agent Management Service (AMS), the Directory Facilitator (DF) and the Message Transport Service (MTS). The distributed nature of the middleware is achieved by using what are called agent containers. This is a specific solution of JADE, which is not part of the FIPA specifications, that allows several instances of the same middleware spread over one or more hosts. The agent mobility provided by JADE, on the contrary of the mobility implementation presented in this section, is between these containers. The JADE middleware architecture is depicted below in Figure 3.

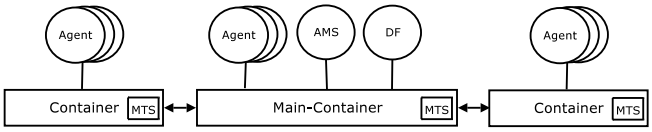

Fig. 3: The JADE agent middleware architecture.

Since JADE v3.2, a new design known as "distributed coordinated filters architecture" is followed. A small kernel and a set of extensible services provide middleware support to agents. The agent software migration architecture presented in Section 3 is implemented as one of these services.

4.2 Implementation of the software architecture

The architecture proposed in Section 3 extends standard JADE migration. Agents can migrate not only within a JADE middleware platform instantiation (between its agent containers), but also between different middleware platform instantiations.

Service level migration When an agent wants to move to another agent middleware platform instantiation, it calls its `doMove()` method with the agent's destination. This call is intercepted by our migration service which notifies the local Agent Mobility Manager (AMM) to start the migration process. Notice that, because of the service architecture of JADE and for compatibility reasons with its own mobility service, the way the migration process is initiated differs from the Figure 2, in which it is initiated by a message issued from the migrating agent to the AMM. This does not represent a problem and does not exclude a future addition of the method shown in the figure. Then, the process described on Section 3.2 is followed by the two involved AMMs. Notice the agent will keep its JADE name in the destination since it is a FIPA compliant name.

Organisation The proposed migration architecture is implemented by the AMM and by a number of additional components: protocols, ontologies, mediators and schedulers.

The AMM is a generic JADE agent which runs several JADE behaviours that offer the proposed architecture functionality. This agent serves migrating agents until the middleware is stopped.

Migration protocols are implemented using FIPA interaction protocols, which in JADE are provided by sets of two behaviours (one acting as an initiator and the other as a participant). *Ontologies* are used to exchange ACL message complex content information by aforementioned protocols. They take advantage of the JADE support for content languages and ontologies. *Mediators* are components that ease the dialog between the AMM and the middleware internal services to support the agent migration process with operations like: serialise, register, start agent, etc. And, finally, *schedulers*, are components that schedule agents that have requested to be migrated to start the migration process.

5 AgentScape implementation

This section describes the second implementation of the migration architecture—the implementation of the migration architecture in AgentScape. The AgentScape middleware, the FIPA messaging service addition and, finally, the migration architecture implementation are described in turn.

5.1 AgentScape middleware

AgentScape middleware [10] (see Figure 4) is a multi-language mobile agent middleware platform designed to support scalable, secure, distributed multi-agent applications. Agents migrate between virtual domains called *locations*. An AgentScape location consists of one or more hosts running the AgentScape middleware, typically within a single administrative domain. Each AgentScape location runs a *location manager* process on one of the hosts. Each host has its own *host manager*.

The AgentScape *kernel* [12] provides low-level secure communication between the higher level middleware processes, between agents and secure agent mobility. *Agent servers* provide language dependent run-time environments for agents, *Web service gateways* support Web service access and monitoring using the SOAP/XML protocol. The kernel is implemented in both Java and C.

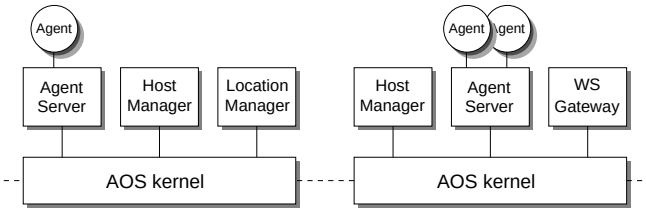

Fig. 4: The AgentScape middleware architecture.

5.2 FIPA Workaround

Due to differences in design objectives, FIPA compliance has never been an AgentScape design objective. AgentScape's main design objective is to support large scale distributed multi-agent applications efficiently, leaving the choice of interaction protocols, languages and naming up to the developer. The option to support FIPA based agents, however, is a design objective. To this end, an implementation of an application level migration architecture is proposed (in addition to the system level protocol AgentScape currently supports) to support the FIPA based migration architecture of Section 2, fulfilling the requirements listed in Section 3.1.

To support ACL messages and the SL content language several classes with this functionality have been ported from the JADE agent platform. Meanwhile, the Agent

Communication Channel (ACC) has been implemented from scratch. Since AgentScape agents does not support FIPA messages by default, a specific component called MessageHandler has been created to allow each agent to send and receive these messages. A directory of agents with their FIPA names has also been included.

5.3 Migration implementation

The following paragraphs describe the implementation details at the application level and the resulting final structure of the architecture.

Application level migration The migration scheme for AgentScape follows the application level procedure as described in Section 3.2 (see also Figure 2).

In the AgentScape migration process, an agent that wants to move to a remote platform, serialises itself, searches for the local Agent Mobility Manager (AMM), comparable to the AMM defined for JADE, and sends a message to it requesting to migrate. The local and remote AMMs take care of the rest of the migration process. Then, the remote AMM creates a new agent and reinstantiates the state of the agent from just before the migration. Since the platform internals have not been modified, it is important to note that the new agent will have a different AgentScape identity, but it will keep the original FIPA name.

Organisation As in JADE the migration architecture is implemented by the AMM. It is composed of several components: protocols, ontologies, mediators, and schedulers.

The AMM is implemented and registered as a standard AgentScape agent. Furthermore, it is registered in the Agent Communication Channel (ACC) to allow communication with other remote AMM agents. Once started, it remains waiting for non-ACL agent messages asking to migrate, and for ACL messages of other AMMs to start the reception of agents.

Migration protocols are implemented by FIPA interaction protocols, which in AgentScape are provided by sets of two classes. One class acts as an initiator and the other as a participant, this last running in background. *Ontologies*, *mediators*, and *schedulers* operate in the same way as its counterparts in the JADE implementation (refer to Section 4.2).

5.4 Open issues

The proposed software migration architecture described in Section 3 leaves some open issues when it is used with middleware that is not compliant with the FIPA specifications. The issues appeared in AgentScape middleware migration implementation and its proposed solutions are presented in the following list:

– **FIPA compliance workaround**: The limited set of FIPA standards needed for the migration architecture has to be added to the middleware. For simplicity reasons, they have been integrated at the middleware application level. This option is the least complex because it can be implemented using static libraries and agents without modifying middleware internals.

- **FIPA agent naming**: Agent local naming scheme should be mapped to the FIPA agent naming scheme and vice versa. In case a direct mapping between the two schemes were not possible, a method based on constant tables linking the two agent names should be developed to do the name translations. The latter method is used in AgentScape.
- **Middleware local agent naming**: Local agent name (the one following the middleware local name scheme specification) will not be maintained in the destination platform, unless it can be mapped directly from the FIPA name. Since this is a FIPA based migration architecture, the only agent information sent is the agent AID (Agent Identification) defined within the FIPA Agent Management specification [7]. In fact, if agents use the FIPA name to refer to them, this is not an issue, because these names are kept. In AgentScape, since the agent identification cannot be directly mapped to a FIPA name, the local agent name cannot be kept.
- **Agent life-cycle**: The agent life-cycle shows the operational state of an agent in a specific moment. By using it, for example, the messaging service can decide to deliver a message to an agent or wait to do this later. When an agent migrates, its life-cycle should be in a specific state (in FIPA the transit state). Then, the agent life-cycle provided by the middleware, using the migration proposed, should be in a state equivalent to the life-cycle defined by FIPA. This might be specially important when, in a nearby future, agents would be able to migrate between different kinds of middleware. In the present implementation, agent local life-cycle is not changed when it is migrating.

These are the issues that have been found implementing the software migration architecture described in Section 2. Some of the solutions have not been implemented as they are not critical, but have to be taken into account for a complete future agent middleware interoperability.

6 Related work

For mobile agents to be deployed on a larger scale, interoperability between platforms needs to be ensured. Ametller *et al.* [1], distinguish several areas of interoperability between agent platforms: executable code, platform, and communication protocols. Milojicic [9] describes process migration issues (not limited to mobile agents), with respect to the executable code area. The Kalong architecture [3], a mobility module used in the Tracy middleware focused on the optimization of migration, has also been used in other middleware [11] but without relaying on agent standards. A number of organisations have initiated the development of standards for mobility at the levels of platform and communication in an attempt to deal with the problem of incompatibility and interoperability with varying degrees of success. One example is the OMG-MASIF [8] specification by the OMG group, which deals with the standardisation of mobile agent APIs. At the level of agent communication, IEEE-FIPA standards have been defined [7].

An early attempt to define standards for mobility [5], proposed in the IEEE-FIPA context, was deprecated due to lack of implementations. The proposal in Section 2 shares the same general aim, supporting multiple migration protocols.

7 Conclusions

This paper presents a software migration architecture based on the model proposed by Cucurull *et al.* [4] for inter-platform agent migration. A list of minimal requirements, incorporating a number of FIPA standards, has shown to be sufficient to implement this architecture on two agent platforms: one fully FIPA compliant, one not.

The JADE middleware, FIPA-compliant, satisfies, by definition, all the architecture requirements. The migration service for JADE extends JADE's standard functionality.[3] The AgentScape middleware, the non-FIPA compliant agent middleware platform, has been extended to fulfill the requirements.[4]

Although the presented work makes it possible to migrate agents between different platforms, the shared ontology for execution interoperability has not yet been defined. This is topic of current research. Once this has been completed, interoperability between different agent platforms will be a fact.

Acknowledgements

Work partially funded by the Spanish project TSI2006-03481, the Catalan project 2005-FI-00752 and the European Social Fund (ESF). The authors are grateful for the support provided by Stichting NLnet `www.nlnet.nl`, Martijn Warnier, and Reinier Timmer.

References

1. J. Ametller, J. Cucurull, R. Martí, G. Navarro, and S. Robles. Enabling mobile agents interoperability through fipa standards. In M. Klusch, M. Rovatsos, and T.R. Payne, editors, *Cooperative Information Agents X*, volume 4149 of *Lecture Notes in Artificial Intelligence*, pages 388–401, Edinburgh, UK, September 2006. CIA 2006, Springer Verlag.
2. F. L. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. Wiley, Chisester, England, 2007.
3. P. Braun and W.R. Rossak. *Mobile Agents: Basic Concepts, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann, San Francisco, CA, 2004.
4. J. Cucurull, R. Martí, S. Robles, J. Borrell, and G. Navarro. FIPA-based interoperable agent mobility. In *Multi-Agent Systems and Applications V*, LNAI. Springer, 2007.
5. FIPA. FIPA agent management support for mobility specification, 2000.
6. FIPA. FIPA request interaction protocol specification, 2002.
7. FIPA. FIPA agent management specification, 2004.
8. OMG Mobile Agent Systems Interoperability Facilities Specification (MASIF).
9. D. S. Milojicic, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou. Process migration. *ACM Computing Surveys*, 32(3):241–299, September 2000.
10. B. J. Overeinder and F. M. T. Brazier. Scalable middleware environment for agent-based Internet applications. In *Proceedings of the Workshop on State-of-the-Art in Scientific Computing (PARA'04)*, pages 675–679, Copenhagen, Denmark, June 2004. Published in Applied Parallel Computing, LNCS 3732, Springer, Berlin, 2006.
11. D. Trinh P. Braun and R. Kowalczyk. Integrating a new mobility service into the JADE agent toolkit. In *MATA*, volume 3744 of *LNCS*, pages 354–363. Springer, 2005.

[3] Available at `http://jipms.sourceforge.net` as development release 1.98.
[4] Available at `www.agentscape.org`.

12. Guido J. van 't Noordende, Benno J. Overeinder, Reinier J. Timmer, Frances M. T. Brazier, and Andrew S. Tanenbaum, *A Common Base for Building Secure Mobile Agent Middleware Systems*, Proceedings of the Workshop on Agent Based Computing IV (ABC'07), pp. 13–25, Wisla, Poland, October 2007.